Optimizing and Exploring System Performance in Compact Processing-in-Memory-based Chips

Peilin Chen, Xiaoxuan Yang

Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA, USA

{peilin, xiaoxuan}@virginia.edu

Abstract—Processing-in-memory (PIM) is a promising computing paradigm to tackle the "memory wall" challenge. However, PIM system-level benefits over traditional von Neumann architecture can be reduced when the memory array cannot fully store all the neural network (NN) weights. The NN size is increasing while the PIM design size cannot scale up accordingly due to area constraints. Therefore, this work targets the system performance optimization and exploration for compact PIM designs. We first analyze the impact of data movement on compact designs. Then, we propose a novel pipeline method that maximizes the reuse of NN weights to improve the throughput and energy efficiency of inference in compact chips. To further boost throughput, we introduce a scheduling algorithm to mitigate the pipeline bubble problem. Moreover, we investigate the trade-off between the network size and system performance for a compact PIM chip. Experimental results show that the proposed algorithm achieves $2.35\times$ and 0.5% improvement in throughput and energy efficiency, respectively. Compared to the area-unlimited design, our compact chip achieves approximately 56.5% of the throughput and 58.6% of the energy efficiency while using only one-third of the chip area, along with $1.3\times$ improvement in area efficiency. Our compact design also outperforms the modern GPU with $4.56\times$ higher throughput and $157\times$ better energy efficiency. Besides, our compact design uses less than 20% of the system energy for data movement as batch size scales up.

I. INTRODUCTION

Convolutional neural networks (CNNs) have gained great success in various applications such as image recognition and autonomous driving [1]–[3]. To improve training and inference accuracy, the complexity and size of CNNs keep increasing. The large-scale parameters of CNNs generate substantial data movement and computation load. Even though prior studies focus on optimizing the computation part, data movement contributes to a significant portion (62.7%) of the total energy consumption [4]. To tackle this challenge, researchers propose processing-in-memory (PIM) architecture [5], [6] to reduce the data movement overhead by storing the data within the computing unit. However, PIM-based chips still face significant challenges when applied to NN inference due to the trade-off between NN model size and available chip area.

Firstly, prior works [7]–[10] overlook the impact of off-chip data movement in their system evaluation of compact (area-limited) PIM designs. As shown in Fig. 1, to put all weights of ResNet152 with 58 million parameters onto SRAM and resistive random access memory (RRAM) designs, the chip area is scaled up to $934.5mm^2$ and $292.7mm^2$. It is impractical to design such large PIM-based chips given the practical area and cost budget [11]. In the PIM processor design [10], we find that the small-scale design, with only 12KB PIM capacity, cannot store all weights of NNs. However, the reporting results only
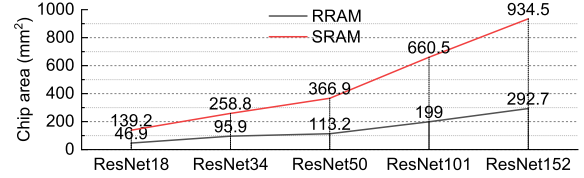

Fig. 1. Chip area required by ResNet NNs to store all weights on SRAM and RRAM array under 32nm process. Such designs are referred to as area-unlimited designs.

consider the energy consumption of all on-chip components, excluding off-chip memory. Integrating off-chip memory for system evaluation is important for understanding the impact of data movement on compact chips.

Secondly, the compact PIM-based chips suffer from poor inference performance compared to impractical area-unlimited designs. Due to the practical assumption of area constraint, compact chips can only load and store a fixed portion of weights each time. It is unavoidable for compact chips to repeatedly erase the current weights and reload other weights. The limited resources hinder compact chips from fully achieving the optimal chip performance. The original pipeline designed for better throughput in area-unlimited chips cannot work effectively on compact chips. Developing techniques specifically tailored to compact PIM-based chips is essential to bridging this performance gap.

Thirdly, for a fixed-scale PIM-based chip, there is a trade-off between the size of deployed CNNs and the performance of leveraging PIM designs. Unavoidable data movement might offset the advantage of compact PIM designs. However, due to PIM advantage in the high parallelism and low energy consumption, there is an upper limit to the size of CNNs that can maintain the performance of PIM-based chips with minimal impact. Therefore, we aim to investigate the trade-off between the deployed NN size and system performance.

Based on these challenges, our work aims to optimize and explore system performance in compact PIM-based chips. The main contributions of our work are as follows.

1) We first analyze the impact of data movement on the performance of compact PIM-based chips. Then, we propose a novel pipeline method to improve the throughput and energy efficiency of inference in compact chips.
2) To further boost throughput, we design a scheduling algorithm to speed up the bottleneck layers dynamically and mitigate the pipeline bubble problem.
3) We explore the maximum size of CNNs that can be deployed onto a given compact PIM-based chip while maintaining its advantage.
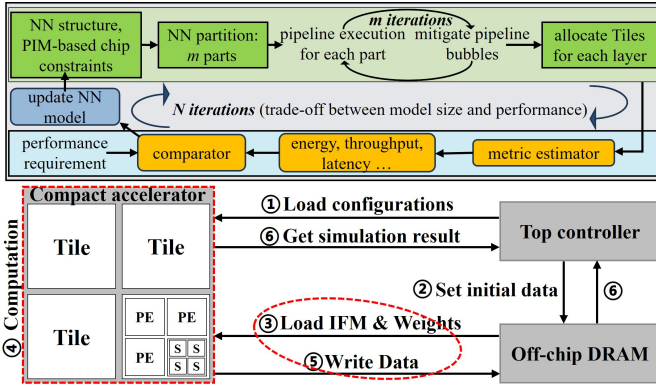
Fig. 2. Overall workflow of our design. PE: Processing Engine. S: Subarray.
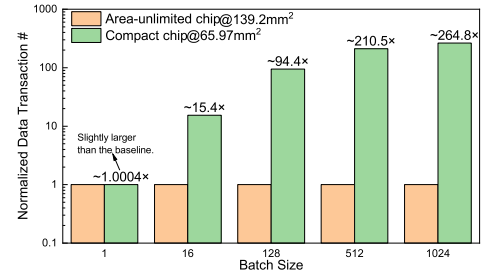

Fig. 3. Normalized data transaction number for different batch sizes between PIM designs and LPDDR5.
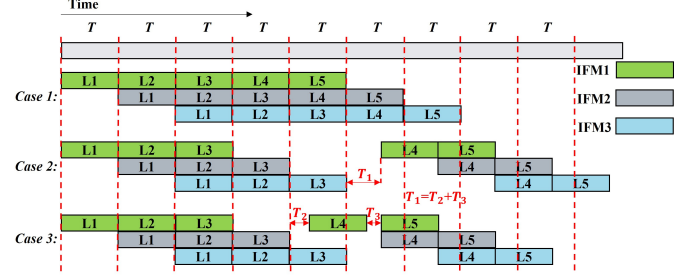

Fig. 4. Pipeline method in area-unlimited designs (case 1) vs. our pipeline method in compact PIM-based chips (case 2 and case 3). In this example, we assume that the NN has five CONV/FC layers (L1~L5).

II. METHODOLOGY

A. Overall workflow

We present the overall architecture and workflow of our design in Fig. 2. We build a compact PIM-based accelerator (denoted in the red dotted box) for evaluation. To ensure fair comparison and accurate benchmarking of system performance, we consider the energy consumption and latency of data movement using different types and specifications of off-chip DRAM, such as LPDDR3 [12], LPDDR4 [13], and LPDDR5 [14]. We precisely record the data movement in steps ③ and ⑤ in the following format: *transaction time*, *transaction type* (write/read), *logical memory address* (32-bit). Moreover, to improve system throughput and reduce the impact of data movement, we propose a novel pipeline method tailored for compact chips and design a scheduling algorithm to mitigate the pipeline bubbles. The search iteration in the upper part covers the NN partition, our proposed designs, resource allocation, metrics evaluation, etc.

B. The Impact of Data Movement on Compact PIM Designs

Data movement can be a severe challenge in compact PIM-based designs. To better understand this issue, we evaluate the data movement of ResNet18 across different batch sizes in the compact chip and the area-unlimited chip that can store all weights of ResNet18, as shown in Fig. 3. When batch size scales up to 1024, the data movement number for the compact chip is $264.8\times$ larger than the area-unlimited design. The compact chip repeatedly reloads both intermediate data and NN weights, while all weights remain fixed in the area-unlimited chip. This is the major factor that significantly offsets the PIM advantage in the compact design. To tackle this challenge, we propose a novel pipeline method tailored to compact chip, design a scheduling algorithm to mitigate the pipeline bubble, and explore the maximum NN size that can deployed in compact chips while maintaining its advantage.

C. Pipeline Method for Compact PIM Designs

Compact PIM-based chips cannot store all the weights of large CNNs. Therefore, for the compact chip, we must design specific principles to partition the NNs. Intuitively,

partitioning the NNs by layers can preserve the NN structure while reducing the storage requirement. To reduce the memory access, we follow partition criteria that allow the chip to map as many layers as possible for each loading process. If the available chip area cannot store one single layer's weights, we would further partition the layer's weight along the input and output channels, which also aligns with work [15]. In summary, our method partitions by layer based on the available storage size and further partitions by channels if necessary.

To boost throughput and energy efficiency in compact PIM-based chips, we propose a novel pipeline method in our design. Fig. 4 shows the difference between the pipeline method in the area-unlimited (where the chip can store all the NN weights) and our proposed pipeline method in compact PIM designs. In Fig. 4, we assume that different layers consume the same time T (T is longer than one cycle) during inference. For case 1 in Fig. 4, we can formulate the time consumption as follows:

$$t(n)_{case1} = L \times T + (n-1)T = (n+L-1)T$$

$$t(perIFM)_{case1} = \frac{t(n)_{case1}}{n} = \frac{(n+L-1)T}{n} \approx T,$$

where L and n denote the layer number and the number of input feature map (IFM) or batch size, and $t(n)$ and $t(perIFM)$ represent the inference time of n IFMs and per IFM, respectively. When there are continuous inputs (or the batch size is large), the chip consumes an average latency of T to complete the inference of each IFM. Compact PIM designs cannot achieve the same optimization effectiveness as in case 1 due to the limited computation resources. However, we propose a novel pipeline method (case 2 and case 3 in

Algorithm 1 Dynamic duplication method (DDM)

Input: $N_{tile}[i]$: Number of *Tile* required by each layer; *E*:
  Extra number of *Tile* after mapping a portion of the NN
  weights onto the chip; *MAX[i]*: Maximum duplication
  number for each layer; *N*: Total number of *Tile*.
Output: *dupNum[i]*: Duplication number of each layer.
 1: Divide NN into *m* parts and partition the ith layer if
    $N_{tile}[i] > N$; Initialize inference time predictor (ITP);
 2: **for** each $i \in m - 1$ **do**
 3:   Initialize the *Flag* and record the minimum *Tile* num-
      bers (min_{tile}) among all layers in this part;
 4:   **while** $E \geq min_{tile}$ **do**
 5:     Update ITP and select bottleneck layer *l*;
 6:     **if** $E \geq N_{tile}[l]$ **then**
 7:       *Flag*=1; $E=E-N_{tile}[l]$; *dupNum[l]=dupNum[l]*+1;
 8:       **if** FC layer **then**
 9:         *dupNum[l]*=1; *Flag*=0;
10:       **else if** *dupNum[l] > MAX[l]* **then**
11:         *dupNum[l]=dupNum[l]*-1; *Flag*=0;
12:       **end if**
13:     **else**
14:       *Flag*=0;
15:     **end if**
16:   **end while**
17: **end for**

Fig. 5. Left: Five NN layers are divided into two parts to be mapped onto the accelerator. Right: Execution process of the NN. WB denotes "Write Back".

Fig. 4) to improve throughput and energy efficiency in compact PIM designs, reducing the performance gap between the area-unlimited designs and compact chips. The *t(perIFM)* for case 2 can be calculated as follows:

$$t(perIFM)_{case2} = \frac{t(n)_{case2}}{n} = \frac{(2n + L - 2)T + T_1}{n},$$

where T_1 denotes the latency for loading the intermediate data and the weights of L4 and L5. The NN in case 2 is divided into two parts. The compact chip performs inference on IFMs in a pipeline manner within each part. By setting a suitable batch size *n* that considers the latency to get the inference result, we can make each part consume the same time as $t(perIFM)_{case1}$. Therefore, $t(perIFM)_{case2}$ is equal to *2T*. Note that *n* in case 2 cannot be infinite because the inference would always remain on the first part and never proceed to the second part. Moreover, we show a special pipeline situation in our design (case 3). Part 2 can ideally be processed *T* earlier if L4 in part 2 requires no more memory capacity than L1 and L2 in part 1, and if the memory requirement for part 2 is larger than that for L1 and L2 in part 1. Different from case 2, case 3 first loads the intermediate data and the weights of L4 into the chip (latency: T_2). After both blue L3 and green L4 finish computation, we begin to load the weights of L5 (latency: T_3). In summary, $t(perIFM)_{case3}$ is equal to $\frac{(2n+L-1)T+T_2+T_3}{n}$.

D. Dynamic Duplication Method

In Section II-C, we assume that each NN layer consumes the same inference time, which is ideal but impractical. The layer with the longest latency will appear as the bottleneck
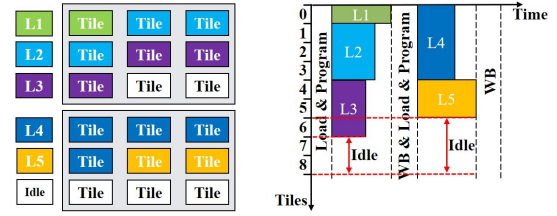
in the pipeline. We need to mitigate the pipeline bubble in order to achieve higher throughput. Therefore, we propose an algorithm to dynamically speed up the bottleneck layers. In our design, we assume that one *Tile* is the minimum computational unit, and mapping more than one layer onto the same *Tile* is not allowed. Due to this assumption, it is feasible to duplicate layers at the *Subarray*, *PE*, and even *Tile* levels. As shown in Fig. 5, there are some idle *Tiles* after mapping selected layers onto the chip. Thus, we can utilize unused *Tiles* to accelerate the bottleneck layers. Algorithm 1 shows the details of the dynamic duplication method (DDM) with our pipeline strategy. We begin by partitioning NN according to the criteria mentioned in Section II-C. Based on the Roofline Model [16], we observe that the inference time of each layer in PIM designs is proportional to the size of the output feature map (OFM) ($O \times O$). Then, we initialize the inference time predictor based on this observation. Following that, we build a *for* loop with *m* iterations (NN is divided into *m* parts) to search for the optimal duplication strategy for each partition part. Note that the maximum duplication value for each layer (*MAX[i]*) is also related to $O \times O$. For example, if *O*=8, we can duplicate this layer up to 64 times, meaning this layer can be computed within one cycle [17]. The *Flag* determines whether to skip a layer that cannot be duplicated. The *While* loop continues if *E* (extra number of *Tiles* after mapping) is greater than the minimum number of *Tiles* required among all layers. Finally, this algorithm returns *dupNum*, which records the duplication number for each layer.

III. EXPERIMENTAL RESULTS AND ANALYSIS

A. Experiment Setup

We build our evaluation framework based on the NeuroSim for PIM-based designs [18] and the DRAMPower [19] for data movement. We design the top controller to support our NN partition strategy, layer duplication, novel pipeline method, and dynamic duplication for bottleneck layers. We select several CNNs to deploy on the compact chip for evaluation, including ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152 [20] for CIFAR-100 dataset [21]. The weights and activations of NN are quantized to 8-bit [22]. We use NeuroSim to benchmark the inference performance of area-unlimited designs. Note that NeuroSim does not explicitly include the data movement of the IFM, weights, and OFM. For fair comparison, both the area-unlimited chip and our compact design use 8Gb 4266MHz 128-bit LPDDR5 as DRAM [14].
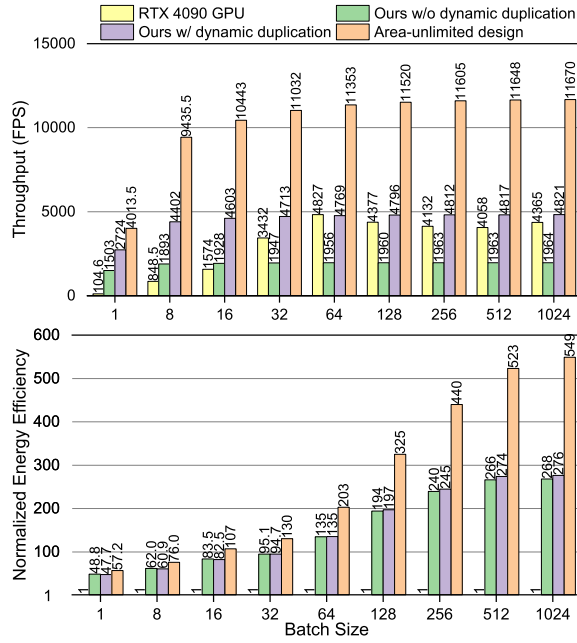
Fig. 6. Evaluating throughput and energy efficiency under different batch sizes: RTX 4090 GPU, our design without and with dynamic duplication method, and area-unlimited chip.
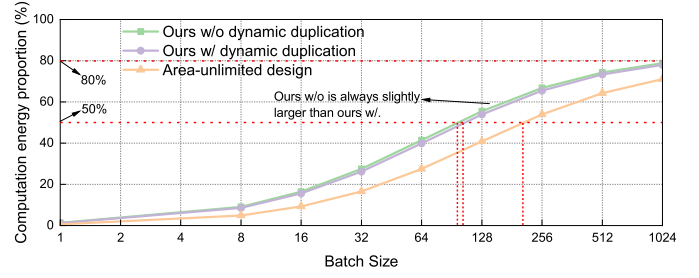

Fig. 7. Computation energy proportion over different batch sizes. Total energy consists of computation energy and off-chip DRAM energy.

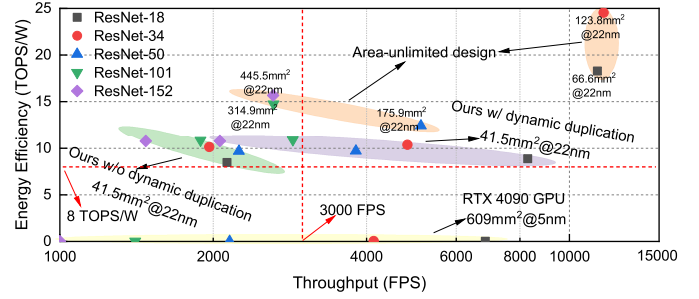Fig. 8. Exploration of maximum NN size deployed in our compact chip.

B. Throughput and Energy Efficiency Improvement

We present the throughput and energy efficiency results for ResNet-34 inference using RTX 4090 GPU, our compact design with an area of $41.5mm^2$, and the area-unlimited chip with an area of $123.8mm^2$ in Fig. 6. Firstly, compared to the RTX 4090 GPU, our design with DDM achieves $4.56\times$ improvement in throughput and $157\times$ improvement in energy efficiency. Secondly, our design with DDM achieves approximately 56.5% of the throughput and 58.6% of the energy efficiency while using only one-third of the chip area compared to the area-unlimited chip. Thirdly, the DDM achieves $2.35\times$ and 0.5% improvement in throughput and energy efficiency compared to the design without DDM. The reason is twofold. On the one hand, we accelerate the bottleneck layers to mitigate the pipeline bubble, thus enabling $2.35\times$ throughput improvement. On the other hand, even though the DDM introduces additional memory access and computation, the benefit of saving leakage energy (due to fewer idle cycles) is more significant. Therefore, energy efficiency gains slightly. Finally, our design with DDM achieves an average of $16.2GOPS/mm^2$ across all batch sizes, which surpasses the area-unlimited design's $12.5GOPS/mm^2$. This highlights our compact design in area efficiency.

C. Data Movement and Energy Breakdown

We further show the proportion of computation energy in the total energy consumption for our design and the area-unlimited chip in Fig. 7. We separate the total energy consumption into two parts: 1) computation energy (which refers to the computation energy of all on-chip components) and 2) off-chip DRAM energy. As the batch size increases, the computation energy occupies more than 50%, and even up to 80%, of the entire system energy. Thus, off-chip DRAM energy does not have a significant impact on system energy efficiency in our design if a suitable batch size is selected for NN inference.

D. Maximum NN Size Deployment in Compact PIM Design

The maximum NN size exploration results are presented in Fig 8. We can observe that the area-unlimited design has obvious advantages over our compact chip due to its sufficient area. However, the DDM narrows the performance gap between our pipeline design and the area-unlimited chip. We focus on our design with dynamic duplication to analyze the exploration of maximum NN size deployment (highlighted by the purple oval). The inference throughput decreases rapidly as the NN grows larger. At the same time, our design has slight fluctuations in energy efficiency but maintains over 8 TOPS/W for all ResNet NNs. For a specific performance requirement (such as energy efficiency $\geq$ 8 TOPS/W and throughput $\geq$ 3000 FPS), it is recommended to deploy ResNet NNs that are smaller than ResNet-101 in our compact chip. Note that this is a rough estimation due to the network's coverage in the current series of experiments. But the maximum size of the ResNet NN that can be deployed onto our compact chip ranges between ResNet-50 (23.7M) and ResNet-101 (42.6M).

IV. CONCLUSION

In this paper, we analyze the impact of data movement on the performance of compact designs and propose a novel pipeline method to improve throughput and energy efficiency. To address the pipeline bubble issue, we develop a scheduling algorithm to speed up the bottleneck layers dynamically. Additionally, we explore and provide insights into the maximum NN size that can be deployed in our compact PIM-based chip.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[3] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[4] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 316–331, 2018.

[5] X. Yang, B. Taylor, A. Wu, Y. Chen, and L. O. Chua, "Research progress on memristor: From synapses to computing systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 5, pp. 1845–1857, 2022.

[6] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L.-Y. Chen, B. Zhang, and P. Deaville, "In-memory computing: Advances and prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, 2019.

[7] Y.-D. Chih, P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naous, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori, *et al.*, "16.4 an 89tops/w and 16.3 tops/mm 2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, pp. 252–254, IEEE, 2021.

[8] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, and J. Chang, "15.3 a 351tops/w and 372.4 gops compute-in-memory sram macro in 7nm finfet cmos for machine-learning applications," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 242–244, IEEE, 2020.

[9] H. Jiang, W. Li, S. Huang, and S. Yu, "A 40nm analog-input adc-free compute-in-memory rram macro with pulse-width modulation between sub-arrays," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pp. 266–267, IEEE, 2022.

[10] F. Tu, Y. Wang, Z. Wu, L. Liang, Y. Ding, B. Kim, L. Liu, S. Wei, Y. Xie, and S. Yin, "Redcim: Reconfigurable digital computing-in memory processor with unified fp/int pipeline for cloud ai acceleration," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 243–255, 2022.

[11] A. Lu, X. Peng, and S. Yu, "Compute-in-rram with limited on-chip resources," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 1–4, IEEE, 2021.

[12] Micron, "Mobile lpddr3 sdram." https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/178b_8-16gb_2c0f_mobile_lpddr3.pdf, 2014.

[13] Micron, "Lpddr4/lpddr4x sdram." https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr4/z19m_embedded_lpddr4_lpddr4x.pdf, 2022.

[14] JEDEC Solid State Technology Association, "Low power double data rate (lpddr) 5/5x." https://www.jedec.org/sites/default/files/docs/JESD209-5C.pdf, 2023.

[15] K. He, I. Chakraborty, C. Wang, and K. Roy, "Design space and memory technology co-exploration for in-memory computing based machine learning accelerators," in *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2022.

[16] S. Williams, "Roofline: An insightful visual performance model for floating-point programs and multicore," *ACM Communications*, p. 16, 2009.

[17] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE international symposium on high performance computer architecture (HPCA)*, pp. 541–552, IEEE, 2017.

[18] P.-Y. Chen, X. Peng, and S. Yu, "Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, 2018.

[19] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power and energy estimation tool." http://www.drampower.info.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[21] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[22] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *International Conference on Learning Representations*, 2018.